A Security Enhancement and Proof for Authentication and Key Agreement (AKA)*

Vladimir Kolesnikov

Alcatel-Lucent Bell Labs Research
600 Mountain Ave. Murray Hill, NJ 07974, USA
kolesnikov@research.bell-labs.com

Abstract

In this work, we consider Authentication and Key Agreement (AKA), a popular client-server Key Exchange (KE) protocol, commonly used in wireless standards (e.g., UMTS), and widely considered for new applications. We discuss natural potential usage scenarios for AKA, attract attention to subtle vulnerabilities, propose a simple and efficient AKA enhancement, and provide its formal proof of security.

The vulnerabilities arise due to the fact that AKA is not a secure KE in the standard cryptographic sense, since Client $\mathcal{C}$ does not contribute randomness to the session key. We argue that AKA remains secure in current deployments where $\mathcal{C}$ is an entity controlled by a *single* tamper-resistant User Identity Module (UIM). However, we also show that AKA is *insecure* if several Client's devices/UIMs share his identity and key.

We show practical applicability and efficiency benefits of such multi-UIM scenarios. As our main contribution, we adapt AKA for this setting, with only the minimal changes, while adhering to AKA design goals, and preserving its advantages and features. Our protocol involves one extra PRFG evaluation and no extra messages. We formally prove security of the resulting protocol. We discuss how our security improvement allows simplification of some of AKA security heuristics, which may make our protocol more efficient and robust than AKA even for the current deployment scenarios.

1 Introduction

This work is positioned at the intersection of engineering security and cryptography. We present a security enhancement of an existing heavily deployed protocol (AKA – Authentication and Key Agreement), and analyze it with formal cryptographic tools. We aim this paper for both crypto and security audiences. Therefore, we use the existing AKA notation and some of the corresponding presentation style and, at the same time, we abstract away non-essential protocol details and follow cryptographic formalisms. The result

*A short version of this paper appears in SCN 2010 [8].

and discussion of the paper is self-contained; standards documents and protocols referenced in this work help put the paper in context for security reader, but are not required for understanding.

Establishment and maintenance of authenticated secure channels is the most used fruit of cryptography today. In particular, wireless and cellular communications critically rely on secure (wired and wireless) channels to exercise control over the network, access, accounting, etc.

In this work, we consider use scenarios (and their security consequences) of one of the most popular wireless protocols – AKA. (Our entire discussion correspondingly applies to AKA-derivative protocols, such as EAP-AKA [4].)

3GPP AKA [2], built as a security enhancement of GSM AKA, is a modern efficient KE protocol, which is based on pre-shared secret key (PSK). It is widely deployed today in GSM and other cellular networks, and is considered for a variety of additional applications.

For logistical reasons (e.g., cellular telephone roaming), there are three players in the protocol: the *user*, user's *home environment* (HE), and the (visited) *serving network* (SN). AKA allows SN to authenticate and exchange keys with the user, without ever being given the user's key. Instead, one-time *authentication vectors* (AV) are issued to SN by the HE. All communication and computation in AKA is very efficient thanks to the use of symmetric-key cryptography. However, because of the complexities of existing SN-to-HE protocols and associated delays, the AVs cannot be retrieved on-demand, are delivered in batches, and, in particular, cannot depend on user-generated nonces[1].

1.1 Related Work

The AKA protocol is proposed and used by the 3rd Generation Partnership Project (3GPP) [2]. EAP-AKA [4], an IETF RFC, is a wrapper around the AKA crypto core, to allow for a standard EAP interface to the protocol.

AKA has been extensively debated and scrutinized by the standards bodies, and, less so, in academic research. 3GPP published a technical report [3] containing an (enhanced) BAN-logic [5] proof of security. However, this proof does not operate with rigorous complexity-theoretic notions, and protocol specification contains occasional imprecise security statements, some of which we note in this paper. We note that no serious security vulnerabilities have been discovered in AKA.

In [13], the authors consider a simple Man-in-the-Middle attack that allows an attacker (a "false base station" in their terminology) to redirect the traffic from a legitimate Serving Network to a Serving Network of his choice. The attack relies on the fact, that AV's issued by User's service provider do not cryptographically bind the ID of the Serving Network to which they are issued. The solution of [13] is to do so. We note that this security issue is different from what we are considering in this work.

[1]Observe the time it takes to switch on a mobile phone first time after landing in a new country versus switching it on for a second time. In the first case, the authenticators are retrieved from HE. In the second case, they are already cached at the visited SN.

1.2 Our Contributions

In this work, we consider the AKA key exchange protocol. We present a simple and intuitive argument of security of the cryptographic core of AKA in the case when Client $\mathcal{C}$ is an entity, controlled by a *single* User Identity Module (UIM), a tamper-resistant hardware token storing the key and performing KE. We identify the logical steps that rely on the single-UIM assumption.

We then argue that in many settings it is natural, convenient and more efficient to allow for multiple UIMs, issued to the same user, to share the secret key. We show that AKA is *insecure* in the above scenario with multiple UIMs sharing the key[2].

Finally, as our main contribution, we show a simple amendment to AKA that closes this vulnerability, and results in a secure key exchange protocol, which we formally prove. The idea of the proposed modification is to require the client to contribute randomness to the resulting session key. We stress that our modification adheres to the design requirements of AKA, and preserves the underlying data flow structure and patterns, and trust model, which is critical in today's deployment scenario. In particular, no extra communication with Home Environment is required, and batching of authenticators AV is possible (actually is simplified and improved). We discuss how this low-cost amendment (one extra PRFG evaluation and no extra messages) adds robustness, allows new usage scenarios, simplifies complicated AKA nonce generation heuristics, prevents UIM cloning attacks, etc.

1.3 Outline

We start, in §2, with presenting in detail the AKA protocol, and argue its security (in the case when each client is controlled by a corresponding *single* UIM). Then, in §3, we discuss the benefits of having several UIMs contain Client $\mathcal{C}$'s identity and credentials, and show AKA vulnerability in this case. We present our enhanced version of AKA KE protocol in §4, discuss its advantages and sketch a proof. We present its formal proof of security in Appendix B. Finally, we conclude in §5.

2 The AKA protocol

In this section we present in detail the cryptographic core of the AKA protocol. See Fig. 1 for the players and flow, and Fig. 2 and Fig. 3 for the precise message description.

Notation. For readability, we will introduce and use standard cryptographic KE notation. However, our presentation is based on [2]; therefore, for the benefit of the security reader, we will include its notation for the ease of cross-reference. In particular, the diagrams are presented with notation of [2]. For reference, we provide a glossary of terms and abbreviations we use and their correspondence.

[2]This scenario is not explicitly disallowed in AKA specification, although the single-UIM setting appears to be implicit in the standards groups. Our discussion of the attacks thus serves to popularize this knowledge and attract attention of potential AKA adopters.

Table 1: Glossary of terms and abbreviations

UIM (User Identity Module) – tamper-proof token	PSK (Pre-shared secret key)
AV (Authentication Vector) – auth. data given to $\mathcal{S}$	SN (Serving Network) – server $\mathcal{S}$
HE (Home Environment) – key server $\mathcal{KS}$	MS (Mobile Station) – client $\mathcal{C}$

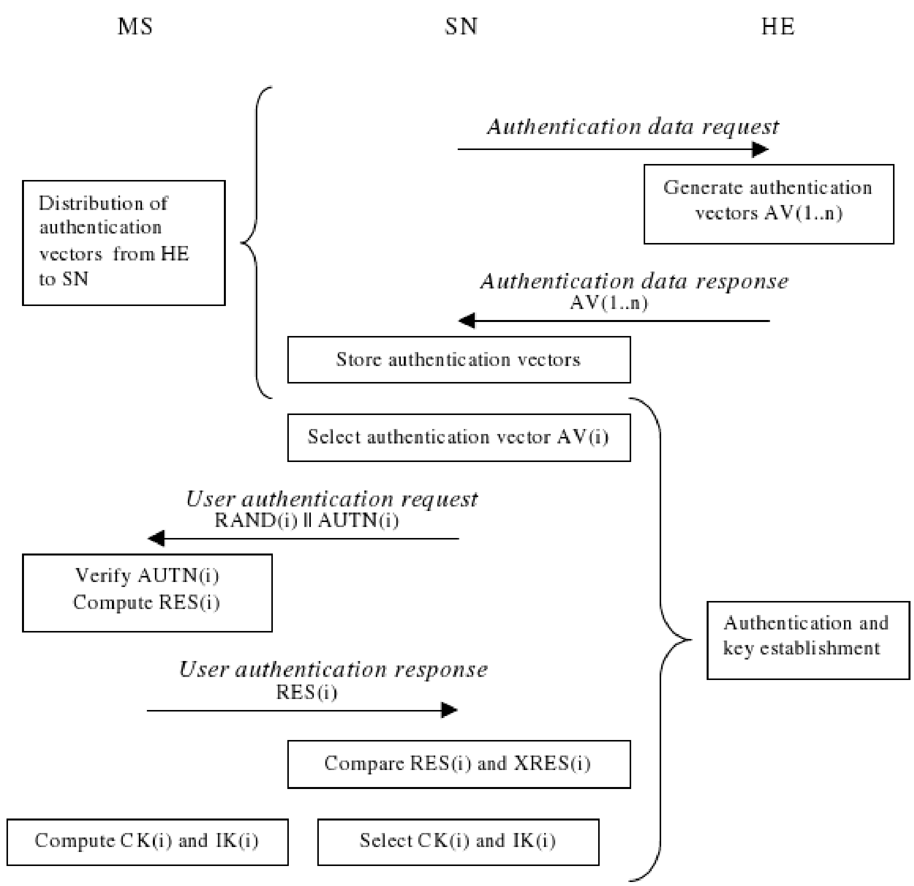

Figure 1: AKA Flow (HELLOs and other details omitted. See §2 for notation.)

2.1 Players, Credentials, and Communication Flow

Players. There are three participants in AKA:

1. Client $\mathcal{C}$ (i.e., mobile phone, called Mobile Station (MS) in [2] and Peer in [4]) is the party initiating the exchange. In the wireless context, $\mathcal{C}$ is usually a MS requesting to be authenticated and granted access to a resource. $\mathcal{C}$ possesses a secret key k which is the basis for authentication.

2. Key Server ($\mathcal{KS}$) (called Home Environment (HE) in [2]). This player is $\mathcal{C}$'s server (in the wireless context, it is the service provider), who had issued $\mathcal{C}$'s key k (usually on a secure User Identity Module (UIM)), and securely stores k. Key Server does not directly exchange keys or authenticate $\mathcal{C}$, but facilitates this process by giving one-time credentials to Server (described next).

3. Server $\mathcal{S}$(called Serving Network (SN) in [2]) is the party who directly interacts with and establishes a secure channel with the Client $\mathcal{C}$. In the wireless context, $\mathcal{S}$ grants $\mathcal{C}$ access to resource (network visited while roaming). As noted, $\mathcal{C}$'s credentials (i.e. key k) is not issued by $\mathcal{S}$, and are unknown to him. Instead, $\mathcal{S}$ receives (possibly batched) one-time authentication vectors AV from Key Server.

Credentials. Key exchange in AKA is based on Pre-shared Secret Key (PSK) k. The key is issued by Key Server to client $\mathcal{C}$; thus both $\mathcal{KS}$ and $\mathcal{C}$ have k. We stress that the server $\mathcal{S}$ *does not* have access to k; instead he receives (as needed, possibly batched) one-time authentication vectors AV from Key Server, which allows $\mathcal{S}$ and $\mathcal{C}$ to mutually authenticate and share a session key.

Trust Relationships. Server and Key Server trust each other and are assumed to have established a secure channel. We do not discuss how authentication vectors are delivered to $\mathcal{S}$; we assume this is done in a timely and secure manner. $\mathcal{S}$ and $\mathcal{C}$, on the other hand, do not *a priori* trust each other; it is the goal of KE to establish a secure channel *only* if parties possess matching credentials.

Data Flow. Upon client's initiation of KE, $\mathcal{S}$ contacts $\mathcal{KS}$ and obtains the authentication vector AV (usually, he would have done this in advance). AV (formally presented in the next section), is a vector, consisting of a challenge for $\mathcal{C}$, an expected response, auxiliary security data, and session keys to be used in case of successful authentication. AV's cannot be reused; to enable multiple logins, $\mathcal{KS}$ sends several AV's, indexed by the sequence number. (It is critical that $\mathcal{S}$ contacts $\mathcal{KS}$ as infrequently as possible, due to unacceptable (minutes-long) delays in current deployments. This imposes a rigid requirement on communication patterns; in particular, Client-generated messages (e.g., nonce) cannot be forwarded to $\mathcal{KS}$.)

$\mathcal{S}$ then sends AV's challenge (consisting of a random nonce RAND and its authenticator AUTN) to $\mathcal{C}$. $\mathcal{C}$ uses AUTN and his key k to confirm that the challenge indeed came from $\mathcal{KS}$. If so, $\mathcal{C}$ computes the session keys, computes and sends back response RES, and is ready to securely communicate with $\mathcal{S}$. $\mathcal{S}$ receives $\mathcal{C}$'s response and compares it with expected response XRES, which was sent by $\mathcal{KS}$ as part of AV. If RES=XRES, $\mathcal{S}$ uses session keys received as part of AV for communication with $\mathcal{C}$.

We do not discuss error handling and other lower level details in this presentation.

2.2 Authentication and Session Key Derivation

In this section, we present in detail the contents of the exchanged messages, and informally argue the security of AKA in the case when each $\mathcal{C}$ is a *single entity*, such as a securely issued UIM.

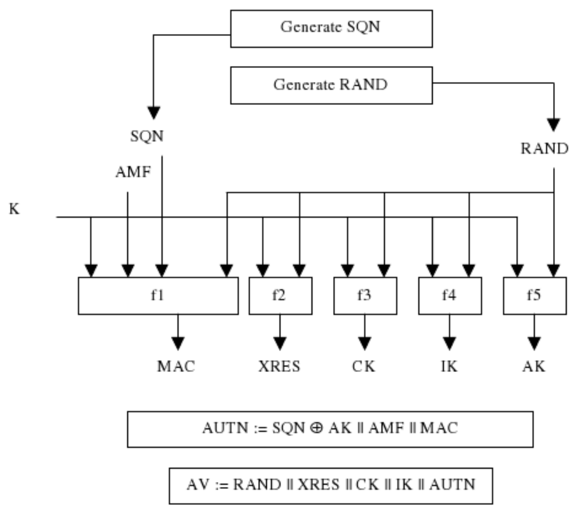

Figure 2: AKA Authentication Vector

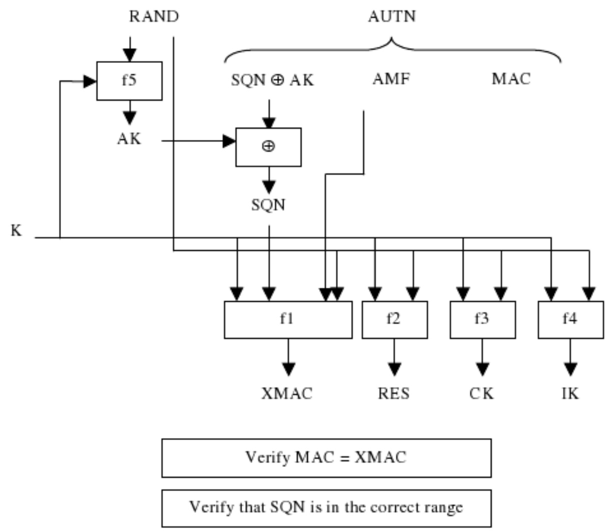

Figure 3: AKA Client Derivation

As mentioned above, we do not discuss the security of the channel between $\mathcal{KS}$ and $\mathcal{S}$; we assume that the adversary cannot read or interfere with this channel. We concentrate on the channel in control of the adversary – the network (or air) connecting $\mathcal{S}$ and $\mathcal{C}$.

Construction of messages referred to in Fig. 1 and discussed in §2.1, are discussed below and graphically shown on Fig. 2 and Fig. 3.

AV is computed as follows.

First, $\mathcal{KS}$ generates a fresh sequence number SQN and a random challenge RAND[3]. SQN is used to keep track of the usage of the one-time AVs, and to help $\mathcal{C}$ detect and reject replays. SQN need not be sequential; [2] discusses (somewhat involved) selection strategies which allow efficient tracking and re-synchronization in case of failures, which is beyond the scope of this paper. AMF, the Authentication Management Field, is used to select technical parameters, such as timeout values; its use is not relevant for our analysis of the core of AKA.

[3]In [2], depending on the occurrence in text, RAND is required to be either "random" or "unpredictable". We note that actually a weaker requirement of *freshness* is sufficient when derivation functions f_i are pseudorandom (e.g., AES).

Then, XRES, CK, IK, AK, AUTN and MAC are derived from PSK k, SQN, RAND, and AMF using message authentication functions f_1, f_2 and key derivation functions $f_3, ..., f_5$. We note that security properties of $f_1, ..., f_5$ are stated insufficiently formally in [2][4]. For simplicity, we assume stronger properties of $f_1, ..., f_5$, namely that they are *pseudorandom*. One can think of these functions as AES evaluated on the argument prefixed with the index of the function, e.g. $f_i(x) = AES(i, x)$. Following the existing notation, we keep the functions $f_1, ..., f_5$ in our presentation.

The derivation proceeds as follows (illustrated on Fig. 2).

- message authentication code MAC
  $\mathsf{MAC}= f_{1_k}(\ \mathsf{SQN}\ ||\ \mathsf{RAND}\ ||\ \mathsf{AMF}\ )$
- an expected response
  $\mathsf{XRES}= f_{2_k}(\mathsf{RAND})$
- cipher key
  $\mathsf{CK}= f_{3_k}(\mathsf{RAND})$
- integrity key
  $\mathsf{IK}= f_{4_k}(\mathsf{RAND})$
- anonymity key
  $\mathsf{AK}= f_{5_k}(\mathsf{RAND})$
- authentication token
  $\mathsf{AUTN} = \mathsf{SQN} \oplus \mathsf{AK}\ ||\ \mathsf{AMF}\ ||\ \mathsf{MAC}$

Here, CK and IK are the keys to be used in the event of successful authentication. The use of AK is optional, and when its generating function f_5 is non-zero, AK is used for hiding the sequence number. We do not consider this option, and set $f_5 = 0$, as it is not relevant to the security of AKA core.

Finally, authentication vector AV is the concatenation
$\mathsf{AV} = \mathsf{RAND}\ ||\ \mathsf{XRES}\ ||\ \mathsf{CK}\ ||\ \mathsf{IK}\ ||\ \mathsf{AUTN}$.

Client's computation. $\mathcal{C}$ receives[5] RAND and $\mathsf{AUTN} = \mathsf{SQN}\ ||\ \mathsf{AMF}\ ||\ \mathsf{MAC}$. (Recall, AMF is not essential for AKA core security). Since $\mathcal{C}$ has possession of the same PSK k that was used in the derivation of AV, $\mathcal{C}$ is able to derive the respective values: response RES, CK, IK, and expected MAC XMAC, as shown on Fig. 3. Then $\mathcal{C}$ verifies that the received MAC equals to expected MAC ($\mathsf{MAC}= \mathsf{XMAC}$) and checks that the sequence number SQN is within the expected (heuristically specified) range. If the checks pass, $\mathcal{C}$ sends his final

[4] In particular, message authentication function f_2, used in the computation of XRES, is allowed to be truncated. We note that, depending on the MAC function, truncation may completely remove its security properties. This is because standard definition of MAC allow for portions of the MAC output to be irrelevant (e.g. always set to 0). Clearly, truncating MAC, leaving only the predictable zeros allows for easy MAC forgery.

[5] Recall we considering the case when $\mathsf{AK} = 0$.

response RES to $\mathcal{S}$, considers that AKA succeeded, and starts using CK and IK that he derived.

Server's computation. $\mathcal{S}$ receives $\mathcal{C}$'s response RES; if $\mathsf{RES} = \mathsf{XRES}$, $\mathcal{S}$ considers that AKA succeeded, and starts using CK and IK that he received from $\mathcal{KS}$.

This completes the description of the cryptographic core of the AKA key exchange.

2.2.1 Security Argument

We present an intuitive argument of security of AKA in the case when each $\mathcal{C}$ is a *single entity*, such as a securely issued UIM. This informal analysis also serves the purpose of identifying the logical steps that use the single-UIM assumption. For simplicity, we assume that each function f_i used is pseudorandom. The analysis of the actual AKA, which relies on MAC, is almost identical.

We proceed with the message flow of AKA, and first consider $\mathcal{C}$ and his actions. Consider the case when $\mathcal{C}$ accepts. (Otherwise $\mathcal{C}$ simply aborts without outputting anything, and, as is easy to see, the adversary $\mathcal{A}$ cannot gain from this. The only information leaked to $\mathcal{A}$ here is the pattern of failures, which he can predict by himself based on the messages he delivers to $\mathcal{C}$.) We first observe that the adversary $\mathcal{A}$ cannot have obtained from $\mathcal{KS}$, $\mathcal{S}$ or $\mathcal{C}$ two vectors AV with the same RAND but different SQN. Since $\mathcal{C}$ accepted, $\mathcal{C}$ must have verified the freshness of SQN (that is that $\mathcal{C}$ had not received AV with this SQN before[6]). We can now see that RAND received by $\mathcal{C}$ is fresh and indeed generated by $\mathcal{KS}$ in the AV with SQN[7]. That is, with overwhelming probability, this RAND had not been used by $\mathcal{KS}$ in other AV's, and it is not replayed[8] or forged by the adversary $\mathcal{A}$. This follows from the unforgeability of PRF (or MAC, if we assume f_1 is a secure MAC, as is done in [2]). Indeed, suppose to the contrary, $\mathcal{C}$ accepted a RAND, SQN pair delivered by $\mathcal{A}$ which was not generated by $\mathcal{KS}$. Then $\mathcal{A}$ must have broken the PRF/MAC property of f_1, by generating a MAC on a new message, since $\mathcal{C}$ must have verified and accepted MAC.

At this point, we know that $\mathcal{S}$ and $\mathcal{C}$ share the same RAND. Thus $\mathsf{RES} = \mathsf{XRES}$ and $\mathcal{S}$ and $\mathcal{C}$ compute the same IK and CK. Further, RAND is fresh[9], that is, $\mathcal{A}$ had not seen $f_{i_k}(\mathsf{RAND})$. Then, even though $\mathcal{A}$ knows RAND, learning anything about $\mathsf{XRES}= f_{2_k}(\mathsf{RAND})$ or $\mathsf{CK}= f_{3_k}(\mathsf{RAND})$ or $\mathsf{IK}= f_{4_k}(\mathsf{RAND})$, implies that $\mathcal{A}$ can break the pseudorandomness properties of f_i (is thus considered impossible).

Now $\mathcal{C}$ computes and sends out $\mathsf{RES} = f_{2_k}(\mathsf{RAND})$. Both $\mathcal{C}$ and $\mathcal{S}$ had agreed on random and unknown to $\mathcal{A}$ session keys. Now, the most damage $\mathcal{A}$ can do is interfere with the delivery of RES to $\mathcal{S}$, preventing the establishment of the secure channel. However this

[6] Here is where the assumption of a single instance of $\mathcal{C}$ is critical. If $\mathcal{C}$ is allowed to be instantiated, e.g., on two devices, freshness cannot be guaranteed.

[7] We are also guaranteed that the AMF received by $\mathcal{C}$ was generated by $\mathcal{KS}$ in the same AV as RAND and SQN. However, we are not addressing the AMF issues in this work, and will not discuss this further.

[8] Here the assumption of a single instance of $\mathcal{C}$ comes in again. If $\mathcal{C}$ is allowed to be instantiated, e.g., on two devices, then replays are possible, since both devices will accept the same sequence number SQN.

[9] Again, provided that $\mathcal{C}$ is instantiated on a *single* device, and the adversary $\mathcal{A}$ thus did not replay the AV.

is not considered a vulnerability in KE, since the same effect can be achieved, e.g., by $\mathcal{A}$ simply jamming the communication channels.

Security Proof. We note that the above security argument can be transformed into a formal proof of security. There are two things that need to be done. First, we need to give a definition of security in the setting with a single-UIM client. Such definition can be derived from existing KE definitions (e.g., [11, 6, 7, 10]) by carefully restricting instance creation in the ideal worlds (simulation-based) or the games (indistinguishability-based) of the definitions. Then, we can transform our argument into a proof based on the proposed definition. We defer these technically involved steps as future work.

3 Multi-UIM setting and AKA vulnerabilities

We start with justifying the setting where the client $\mathcal{C}$ would possess several devices provisioned with the same PSK (let's call it the *multi-UIM setting*).

Today, service providers aim to engage the customers as much as possible, and offer bundled services, such as triple play (phone, internet, TV). It is easy to imagine that the next step is to allow customers to have a number of devices, such as laptop, phone, camera, etc., to use the service.

We first observe that it is convenient to decouple the user (i.e. subscriber) from the device (as done, e.g., by GSM and the WiMAX Forum [1]). This allows attractive business scenarios, where one customer may have several subscriptions, and use several devices (e.g., laptop, phone, camera, etc. in case of WiMAX). The convenience factor for the user is that the devices are all linked in one plan, and the service is not linked to the device, but rather to the subscriber identity. A person should be able, e.g., to borrow or buy a new device, plug in his UIM card, and use it. The devices often need to be "swappable", as is done today with the removable UIM cards. The UIM cards should not be tied to the type of service (TV, phone, etc.), but should be interchangeable and mainly serve to authenticate the customer.

In cases as above, it may be convenient (although, of course, not absolutely necessary) to employ multi-UIMs (i.e., UIMs issued to the same $\mathcal{C}$ and initialized with the same PSK). Issuing several *identical* UIM cards to the customer is convenient to the user and the service provider, and it brings significant efficiency gains, as described next.

Convenience and efficiency gains with multi-UIM. Clearly, this ensures the swappability as described above, and the associated convenience for the customer. We note that this feature can be emulated by issuing UIMs with different keys, and $\mathcal{KS}$ linking them to the customer's account and keeping track of all the keys. This is a feasible replacement, which, however, comes at a cost. Firstly, $\mathcal{KS}$ must keep track of and *securely store* a much larger number of keys (a factor of 5 to 10 in the near future, depending on the average number of devices per user). Secure storage is expensive, and this is a significant penalty. Further, generating and delivering batched AV is much more convenient with the multi-UIMs, since any of the AV's generated for the customer would work. In contrast, if each of customer's UIMs has a separate key, a separate AV must be requested and delivered to $\mathcal{S}$, causing latencies, additional network load, and increased $\mathcal{S}$ storage.

3.1 Multi-UIM vulnerabilities of AKA

As already noted, in AKA, $\mathcal{C}$ does not contribute randomness to the resulting session key, and instead relies on the freshness of SQN, which can only be guaranteed if PSK k is stored securely on a UIM and only a *single* UIM is issued per customer identity. At the same time, as discussed above, employing multiple UIMs keyed with the same PSK k is often convenient and more efficient.

Because AKA Client contributes no randomness, the session key is entirely determined by the RAND (and its authenticator AUTN) sent by server. We now show two simple attacks on AKA deployment in the multi-UIM scenario, i.e. if there are two instances (e.g., devices) of $\mathcal{C}$ using the same PSK.

Denote by $\mathcal{C}_1$ and $\mathcal{C}_2$ the two instances/devices of $\mathcal{C}$ sharing the same PSK k.

Attack scenario 1. $\mathcal{C}_1$ and $\mathcal{C}_2$ both wish to connect to the network. $\mathcal{C}_1$ initiates the exchange, and, as prescribed by AKA, $\mathcal{S}$ sends RAND, AUTN to $\mathcal{C}_1$, which $\mathcal{C}_1$ receives, and the adversary $\mathcal{A}$ overhears. $\mathcal{C}_2$ initiates the exchange, which $\mathcal{A}$ blocks from $\mathcal{S}$. Instead, $\mathcal{A}$ replays the RAND, AUTN message to $\mathcal{C}_2$. At this time, both devices $\mathcal{C}_1$ and $\mathcal{C}_2$ derive the same session keys CK,IK, but they both think they are talking to $\mathcal{S}$. Carefully forwarding (presumably secured) messages sent between the two devices and the server may allow $\mathcal{A}$ to create unintended transactions on $\mathcal{C}$'s account. For example, if the transaction performed on $\mathcal{C}_1$ involves a debit on the account maintained by $\mathcal{C}_1$'s UIM, the adversary $\mathcal{A}$ replaying this transaction to $\mathcal{C}_2$ (possible, since $\mathcal{C}_2$ has the same session key as $\mathcal{C}_1$) may effect a corresponding debit on $\mathcal{C}_2$'s UIM – clearly an unintended transaction and a successful attack.

Attack scenario 2. The above attack is strengthened if the adversary $\mathcal{A}$ borrows (or captures or remotely compromises) one of $\mathcal{C}$'s devices containing the PSK k (say, $\mathcal{C}_2$). By good engineering practices, PSK k is securely stored on UIM, which it never leaves. Thus, with k unavailable to $\mathcal{A}$ a secure system should guarantee that the compromised device $\mathcal{C}_2$ should not help $\mathcal{A}$ compromise other devices (e.g., $\mathcal{C}_1$) or their sessions.

Not so with using AKA in this scenario. Indeed, the session keys produced by the UIM are exported into the main memory of the device, which $\mathcal{A}$ can exploit by performing an attack similar to described above. $\mathcal{A}$ simply overhears RAND, AUTN destined to $\mathcal{C}_1$, and forwards it to the UIM in his control. As prescribed by AKA, $\mathcal{A}$'s UIM will generate and export to the device session keys CK,IK, which are the same keys generated on $\mathcal{C}_1$. Clearly, the adversary $\mathcal{A}$ is now in control of the (presumably) secure session established between $\mathcal{S}$ and $\mathcal{C}_1$.

We stress that this attack is especially dangerous if user's devices have different degrees of confidence. Then, the attacker, by remotely hacking an unimportant and weakly protected device, such as child's laptop, may gain access to a high-confidence device on the same account, such as parent's smart phone.

4 Secure Multi-UIM AKA

In this section we present our main contribution – a simple and efficient AKA security enhancement for the multi-UIM case. We formally prove security of our protocol; in particular, we close the multi-UIM vulnerability. The idea of the enhancement is to have the client $\mathcal{C}$'s UIM(s) generate and contribute their randomness to the session key, in a way that preserves AKA message flow.

The most natural (and sufficient!) method to do it is to use the already established CK,IK as *intermediate* keys only, and derive the session keys based on randomness sampled by UIM. For example the actual session keys could be $\mathsf{CK'} = f_{\mathsf{CK}}(\mathsf{RANDC})$, $\mathsf{IK'} = f_{\mathsf{IK}}(\mathsf{RANDC})$, where $\mathsf{RANDC} \in_R \{0,1\}^n$ is sampled by UIM. Now, $\mathcal{C}$ could simply send RANDC to $\mathcal{S}$ to enable server-side derivation. We note that it is critical that UIM never exports to the device anything other than the final session keys.

Intuitively, security now holds since only the parties who possess CK,IK are able to derive the session key by evaluating the prescribed derivation function keyed with CK,IK on argument RANDC. These parties are the (authorized) $\mathcal{C}$ (his UIM sampled RANDC and evaluated on it), and $\mathcal{S}$, who is given CK,IK by $\mathcal{KS}$. The adversary $\mathcal{A}$ is not able to compute session keys $\mathsf{CK'}$,$\mathsf{IK'}$, even if he compromised $\mathcal{C}$'s additional devices, since UIMs of these devices only evaluate derivation functions on the arguments they (UIM's) sample themselves.

Next, we present our protocol in detail, and give intuition for its security. We present a formal proof of security in the Appendix.

4.1 The formal Multi-AKA Protocol

The protocol, informally presented above, demonstrates the step that we need to take to achieve security in the multi-UIM setting. This protocol can be naturally simplified and optimized, which we do in this section. Importantly, we keep the message structure, efficiency, and features of the original AKA protocol.

We observe that there is no need to include two derived keys (CK,IK) into AV. One derivation key KD computed from PSK k and RAND is sufficient to derive CK and IK to be used in the (now secure) session. One simple optimization we perform is using just one key KD in place of CK,IK in AV.

Let $f_1, F, F1, F2$ be pseudorandom function generators, such as AES[10].

Protocol 1 *Multi-AKA*

1. *As follows from the above discussion and examples, in Multi-AKA, $\mathcal{KS}$ will not send the session keys CK,IK to $\mathcal{S}$ in the AV. Instead, $\mathcal{KS}$ will send to $\mathcal{S}$ the session derivation key KD, which is derived from PSK k and the AV's randomness RAND:* $\mathsf{KD} = F_k(\mathsf{RAND})$. MAC *is computed by $\mathcal{KS}$ as before:* $\mathsf{MAC}= f_{1_k}(\mathsf{SQN} \mid\mid \mathsf{RAND} \mid\mid \mathsf{AMF}$

[10] As already discussed above, these functions can either be different functions, or the same function such as AES. In the latter case, we must ensure differentiation of the evaluation of the functions, e.g. by prepending the (AES) argument with a function-unique prefix.

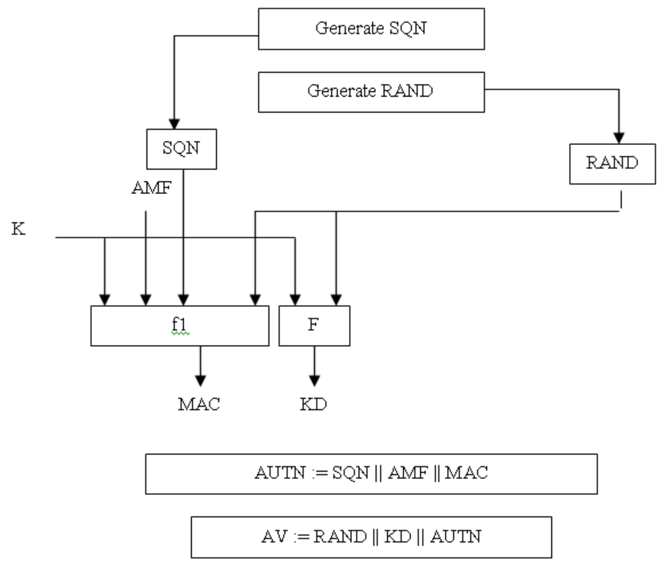

Figure 4: Multi-AKA Authentication vector

). XRES *is omitted since* $\mathcal{S}$ *will compute it itself based on* KD *and* $\mathcal{C}$*'s randomness.* AUTN *is computed as before (*AUTN = SQN || AMF || MAC*)*[11]*, and* AV *is set to* AV = RAND || KD || AUTN*. This step is illustrated on Fig. 4.*

2. *Upon* $\mathcal{C}$*'s request for authentication,* $\mathcal{S}$ *forwards* RAND *and* AUTN *to* $\mathcal{C}$*.* $\mathcal{C}$ *verifies* MAC *as in AKA, and, if successful, proceeds as follows. First, it computes his version of the derivation key* KD = F_k(RAND)*. Then* $\mathcal{C}$ *samples random* RANDC *and computes* RES = F_{KD}(RANDC)*.* $\mathcal{C}$ *also computes* CK= $F1_{\mathsf{KD}}$(RANDC)*, and* IK= $F2_{\mathsf{KD}}$(RANDC)*. Then* $\mathcal{C}$ *sends* RANDC *and* RES *to* $\mathcal{S}$ *and is ready to communicate over a channel secured by session keys* CK,IK*. This step is illustrated on Fig. 5.*

3. *Upon receipt from* $\mathcal{C}$ *the pair* RANDC *and* RES*,* $\mathcal{S}$ *proceeds as follows. He first computes* XRES= F_{KD}(RANDC)*, and checks that* XRES= RES*. If so,* $\mathcal{S}$ *derives (and uses) the session keys* CK= $F1_{\mathsf{KD}}$(RANDC)*, and* IK= $F2_{\mathsf{KD}}$(RANDC)*. This step is illustrated on Fig. 6.*

Remark: All actions of $\mathcal{C}$ *are performed on a UIM, and only the resulting keys* CK,IK *are exported outside of the UIM.*

4.2 Security Argument and Claim

The security proof of Multi-AKA is actually simpler than that of AKA presented above in §2.2.1. The main reason is that with both $\mathcal{S}$ and $\mathcal{C}$ contributing randomness to session keys, message replays don't matter, since they *always* result in the generation of *unrelated* keys, which cannot help the adversary $\mathcal{A}$ in attacking other sessions.

[11] Recall, we set AK = 0.

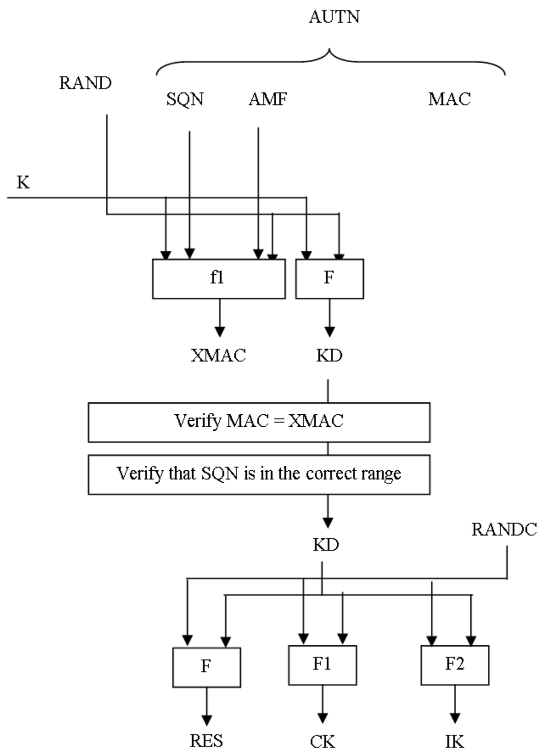

Figure 5: Multi-AKA Client Computation

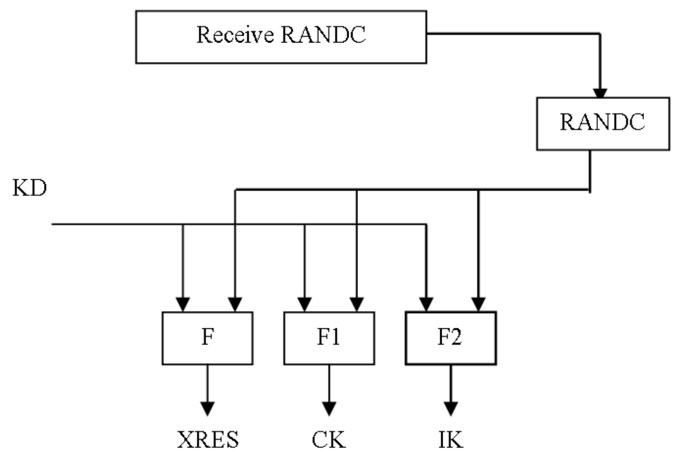

Figure 6: Multi-AKA Server Computation

We proceed with the message flow of Multi-AKA, and first consider $\mathcal{C}$ and his actions. Consider the case when $\mathcal{C}$ accepts (otherwise $\mathcal{C}$ simply aborts without outputting anything, and, as is easy to see, the adversary $\mathcal{A}$ cannot gain from this. The only information leaked to $\mathcal{A}$ here is the pattern of failures, which he can predict by himself based on the messages he delivers to $\mathcal{C}$.) We first observe that $\mathcal{S}$ cannot have received from $\mathcal{KS}$ two vectors AV with the same RAND but different SQN (this is because the probability of collision of two randomly sampled RAND values is negligible). Therefore, there cannot be two instances of $\mathcal{S}$ that generated the same KD. We note that $\mathcal{A}$ can cause different instances of $\mathcal{C}$ (e.g., $\mathcal{C}_1$ and $\mathcal{C}_2$) to generate the same KD simply by replaying $\mathcal{S}$'s message. However, it is easy to see that even in the case when $\mathcal{A}$ controls one of client's instances (say, $\mathcal{C}_1$), the session keys CK,IK output by $\mathcal{C}_2$ cannot be predicted by $\mathcal{A}$, due to the security (namely, pseudorandomness) properties of the function used to generate the keys. This also means that $\mathcal{A}$ cannot arrange for secure channels between unintended parties (e.g., as in our attacks in §3.1, where $\mathcal{S}$ shared session key with two clients). Here, it is critical that UIMs do not export intermediate keys (KD) into the main memory of the device[12].

Similarly, $\mathcal{A}$ cannot learn server keys or mismatch server secure channels (e.g., as in our attacks in §3.1). This is because, firstly, intermediate keys KD generated by any player are

[12]If in fact KD was exported, then $\mathcal{A}$ could trivially compute the session key established by $\mathcal{C}_2$: $\mathsf{CK}= F1_{\mathsf{KD}}(\mathsf{RANDC})$.

unpredictable to $\mathcal{A}$. Further, as noted above, no two servers use the same KD, and thus replaying client's response to $\mathcal{S}$ will cause a verification failure.

Above discussion leads to the following theorem:

Theorem 1 *Protocol 1 (Multi-AKA) is a secure key exchange protocol. In particular, it remains secure if the adversary $\mathcal{A}$ corrupts one or more of $\mathcal{C}$'s instances (devices), but does not get access inside the UIM executing the protocol.*

We approach the proof of Theorem 1 as follows. First, we further abstract away the non-essential messages and concepts and leave only the core message exchange. Then we recall a KE definition (derived from a more general definition of [10]), and prove security of the abstracted protocol with respect to that definition. Finally, it is easy to see that this implies the statement of Theorem 1.

We present a formal proof of security in the Appendix.

4.3 Practical Implications and Considerations

We mention several practical implications of our proposed protocol. First, as can be seen by direct inspection of our abstracted protocol (Appendix §B), we do not use SQN at all. This means that (at least some of) the complicated heuristics associated with the state maintenance can be avoided or simplified. Further, the persistent AKA problem of UIM cloning goes away in the following sense. Of course, a cloned UIM would be able to access the resource as well as the legitimate UIM, however, he would not be able to mount man-in-the-middle attacks on the legitimate UIM's connection.

Revocation. At the first glance, the issue of revocation would become significantly more complex, since all of $\mathcal{C}$'s UIMs share the same key. However, this does not appear to pose any problems, for the following reasons. One revocation solution can use broadcast encryption techniques, and, upon $\mathcal{C}$'s request, simply update the keys of legitimate UIMs, while excluding the stolen ones. We note, that this would require each UIM storing a small number of keys, depending on the maximum number of user's devices. Other solutions could use out-of-band secure channel (e.g., telephone conversation with an agent or a separately protected web control page) to obtain PIN(s) required to refresh the keys of the authorized UIMs. During refresh, short authenticated string (SAS)-based techniques [12] may prove helpful.

4.4 Performance

As it is easy to see, the costs of added security that we propose are negligible. The final tally would depend on the exact instantiations, but our core protocol has only one (1) additional PRFG evaluation by $\mathcal{S}$, as he now needs to compute XRES himself (however, this offloads the corresponding operation on $\mathcal{KS}$). We do not further calculate these costs, since they are negligible compared to AKA communication costs, in terms of energy consumption and time.

5 Conclusion

We considered a widely used AKA protocol and the issue of its reliance on the uniqueness of the tamper-resistant module, UIM, holding user's key. We presented the intuition for security of AKA in this setting. We noted that issuing multiple UIMs to the user, all of which hold the same user's key is appealing for UIM interchangeability, and allows for protocol efficiency improvement, such as better reuse of authenticating data and reduction of required secure storage.

As AKA turns out to be insecure in this setting, we presented a security enhancement of AKA, along with a formal proof of security. Our protocol has negligible performance premium, and is more robust than AKA, while adhering to AKA design goals. Its robustness offers avenues for simplifications of AKA heuristics. We believe our protocol may be a worthy upgrade for AKA and a candidate for more general scenarios.

Acknowledgements. I would like to thank anonymous reviewers of this paper for many valuable comments, and Simon Mizikovsky for our initial discussion about AKA.

References

[1] WiMAX Forum. `http://www.wimaxforum.org/`.

[2] 3rd Generation Partnership Project. 3GPP Technical Specification 3GPP TS 33.102 V7.1.0: "Technical Specification Group Services and System Aspects; 3G Security; Security Architecture (Release 7)", December 2006.

[3] 3rd Generation Partnership Project. 3GPP Techical Report TR 33.902: "Formal Analysis of the 3G Authentication Protocol", March 2001. `http://www.3gpp.org/ftp/Specs/html-info/33902.htm`.

[4] J. Arkko and H. Haverinen. IETF Network Working Group: Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). RFC 4187, January 2006. `http://tools.ietf.org/html/rfc4187`.

[5] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[6] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474, London, UK, 2001. Springer.

[7] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351, London, UK, 2002. Springer.

[8] Vladimir Kolesnikov. A security enhancement and proof for Authentication and Key Agreement (AKA). In *7th Conference on Security and Cryptography for Networks, SCN 2010*, 2010.

[9] Vladimir Kolesnikov and Charles Rackoff. Key exchange using passwords and long keys. In *Theory of Cryptography, TCC 2006*, volume 3876 of *LNCS*, pages 100–119. Springer, 2006.

[10] Vladimir Kolesnikov and Charles Rackoff. Password mistyping in two-factor-authenticated key exchange. In *ICALP (2)*, pages 702–714, 2008.

[11] Victor Shoup. On formal models for secure key exchange. Technical Report RZ 3120 (#93166), IBM, 1999.

[12] Serge Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Advances in Cryptology – CRYPTO 2005*, volume 3621, pages 309–326, 2005.

[13] Muxiang Zhang and Yuguang Fang. Security analysis and enhancements of 3gpp authentication and key agreement protocol. *IEEE Transactions on Wireless Communications*, 4:734–742, 2005.

A Definition of Security of KE

In this section, we present the definition of security we use in our proofs. This definition is a natural adaptation (actually, simplification) of the definitions of Kolesnikov and Rackoff [9, 10]. The latter definitions consider a substantially more general setting of multi-factor authenticated KE, where parties possess both long and short keys (e.g. PSK and passwords). The definitions of [9, 10] have the graceful degradation property, that is, a compromise of some of the keys results in the security level accorded by the remaining key(s). Naturally (also indirectly implied in [9, 10]), omitting the use of short keys results in the definition for our setting.

While one can use one of several KE definitions, we found game-based definition to be the simplest to use. For completeness and the formalization of discussion, we now present the adapted definition that we will use.

We denote by C_i^S the i-th instance of client C who wants to talk to (some instance of) server S. S_j^C is defined symmetrically.

KE definitions rely on the notion of *partners* to specify what constitutes an attack. Informally, two instances of players are partners, if they establish a secure channel. Syntactically, we define partners as follows.

Definition 1 *We say that an instance C_i^S of a client C and an instance S_j^C of a server S are* partners*, if they have output the same session id sid.*

Session id *sid* is an additional (somewhat artificial) output of KE, which need not be used in real execution, but which is needed for syntactic manipulations in the proof. We omit the detailed discussion of the need of *sid* and refer the interested reader to literature, e.g., [9, 10] for additional information.

We start by presenting the KE game, which model attacks of a real-life adversary. Recall, we do not address the security aspects of the server (SN in AKA) receiving the authenticating credentials from the key server $\mathcal{KS}$; we assume this is done in a secure manner, e.g. using pre-established channels.

Game KE. *An honest server S is created. Adv then runs players by executing steps 1-6 multiple times, in any order:*

1. *Adv creates an honest client C. C is registered with S, and a randomly chosen key is set up and associated with C. Only one honest client can be created.*

2. *Adv creates a corrupt client B^i. A randomly chosen key is set up and associated with B_i.*

3. *Adv creates an instance C_i of the honest client C. C_i is given (secretly from Adv) as input the key associated with C.*

4. *Adv creates an instance S_j of the honest server S. S_j is given (secretly from Adv) as input the partner client's key*[13].

5. *Adv delivers a message m to an honest party instance. The instance immediately responds with a reply (by giving it to Adv) and/or, terminates and outputs the result (a sid and either the session key, or the* failure symbol $\perp$*) according to the protocol. Adv learns only the sid part of the output.*

6. *Adv "opens" any successfully completed and checked honest instance – then he is given the session key output of that instance.*

Then Adv asks for a challenge on an instance of an honest player.

The challenge of instance S_j^C of the server S is handled as follows. S_j^C, who has been instantiated to talk to the honest client C, must have completed and output a session key. The challenge is, equiprobably, either the key output by S_j^C or a random string of the same length. Adv must not have opened S_j^C or a partner of S_j^C, and is not allowed to do it in the future.

The challenge of instance C_j^S of the client C is handled symmetrically.

Then Adv continues to run the game as before (execute steps 2-6). Finally, Adv outputs a single bit b which denotes Adv's guess at whether the challenge string was random. Adv wins if he makes a correct guess, and loses otherwise. Adv cannot "withdraw" from a challenge, and must produce his guess.

The above game is almost sufficient for security definition. The only remaining technical aspect is the enforcement of non-triviality. We need to prevent improper partnering (e.g. players unnecessarily outputting same *sid*). Recall, *Adv* is not allowed to challenge parties whose partner has been opened; SID ensures that *Adv* is not unfairly restricted. We handle this by introducing the following game

[13] Other authenticating credentials may be given instead, such as KD in the AKA setting.

Game SID *is derived from game KE by adjusting the win condition. Adv does not ask for (nor answers) the challenge. Adv wins if any two honest partners output different session keys.*

Note, SID allows for one (or both) of the partners to output a failure symbol. *Adv* only wins if two successfully completed parties output different session keys.

We are now ready to present the definition.

Definition 2 *We say that a key exchange protocol* Π *is secure, if for every polytime adversaries* Adv_1, Adv_{sid} *playing games KE and SID, their probabilities of winning (over the randomness used by the adversaries, all players and generation algorithms) is at most only negligibly (in security parameter* n*) better than:*

- $1/2$*, for KE,*
- 0*, for SID.*

B Theorem Statement and Proof of Security of Protocol 1

We now formally state and prove the security theorem of our proposed Protocol 1.

For simplicity, we consider only the core of the protocols, leaving out the non-security-essential messages, and slightly rearrange the PRFG calls in a natural way. It is clear that the following Protocol 2 represents Protocol 1, and the latter is secure if so is the former. We will thus consider and prove security of the following protocol.

Protocol 2 *(Essential message exchange of Protocol 1)*

S^C		C^S
given $r \in_R \{0,1\}^n$, $\mathsf{KD} = F_k(1,r)$, $\mathsf{MAC} = F_k(2,r)$		*given* k
	r,$\mathsf{MAC}\rightarrow$	
		verify $\mathsf{MAC} = F_k(2,r)$ *if fail, abort* *choose* $r_c \in_R \{0,1\}^n$
	$\leftarrow r_c, F_{\mathsf{KD}}(3,r_c)$	
verify $F_{\mathsf{KD}}(3,r_c)$ *if fail, abort*		
set $sk = F_{\mathsf{KD}}(4,r_c)$ *set* $sid = (r,r_c)$, *Output* (sk,sid)		*set* $sk = F_{\mathsf{KD}}(4,r_c)$ *set* $sid = (r,r_c)$, *Output* (sk,sid)

Theorem 2 *Let* F *be a PRFG. Then, protocol 2 is a secure KE protocol, according to the Definition 2.*

As mentioned above, the proof of Theorem 2 implies the security of Protocol 1.

Our proof will consist of several steps. First, we introduce a game G (see below), a simplification of the KE game, and show (Lemma 1) that any *Adv* winning KE implies a corresponding distinguisher for G. Then we show that existence of such a distinguisher for G implies an attack on the employed PRFG. This implies that KE game cannot be won by polytime *Adv* if PRFG is secure. Finally we observe that such *Adv* cannot win the game SID neither. Indeed, in our protocol, partners never output different keys (since the session key is determined by sid).

Game G.

Game is parameterized by the security parameter n, and is played with a distinguisher Dist. Dist proceeds by executing the following Step multiple times:

1. *Dist queries the* PRFG oracle $O_F(i, j, r) = F_{k_i}(j, r)$, *where the random key k_i is chosen by the game, remembered for potential future calls to O_F, and not revealed to Dist. Here $r \in \{0,1\}^n$, integer i and $j \in \{1, 2\}$ are chosen by Dist.*

Then Dist chooses i and $r_0 \in \{0,1\}^n$ and queries the challenge oracle $O_C(i, r_0)$. *O_C produces a challenge as follows: it randomly chooses a bit b and a string $\rho \in_R \{0,1\}^n$. Then $O_C(i, r_0) = F_{k_i}(4, r_0)$ if $b = 0$, and $O_C(i, r_0) = \rho$ if $b = 1$. Dist is not allowed to query $O_C(i, r_0)$, if he queried $O_F(i, 4, r_0)$.*

Then, Dist continues running the above Step 1 multiple times, with the exception that he is not allowed to query $O_F(i, 4, r_0)$.

Finally, Dist outputs a bit b'. Dist wins if $b = b'$.

Lemma 1 *Suppose there exists an adversary Adv that wins KE with probability non-negligibly greater than $1/2$. Then there exists Dist winning the game G with probability non-negligibly greater than $1/2$, where G is run with the same PRFG F as game KE.*

Proof: We present a distinguisher *Dist* and show that it wins G essentially whenever *Adv* wins the KE game. *Dist* simulates an environment (i.e. KE players and their actions), in which he runs *Adv*, answers *Adv*'s queries and uses *Adv*'s decisions to make decisions in G. We say "*Dist* stops", meaning "*Dist* finishes processing *Adv*'s request and returns control to *Adv*", and "*Dist* sends (outputs) m", meaning "*Dist* simulates the given player sending (outputting) m, by giving m to *Adv*".

Dist starts up *Adv*. *Dist* then runs *Adv* and satisfies its requests for information as follows. Note that a client C must have been created to create its instances C_i or server instances S_j^C. We also note that while we carefully handle messages and responses associated with the honest client – this is how we reduce the KE game to the simpler game G – *Dist* simply executes the protocol to handle messages associated with corrupt clients. This is because the keys for each client are independent, and each instance (both of server or client) only possesses the keys relevant *only* to the the communication with its partner instance. That is, for example, server instance talking to client B will not know anything about C's keys, and thus cannot possibly help in attacking C. (We note that a similar issue is addressed in [9]; we refer the interested reader there for more intuition and justification.)

Variables notation. In the following, we will use indexed variables. We will adhere to the following notation. Messages generated by S (i.e. by $Dist$ on behalf of S) will be indexed with letter j. Messages generated by C will be indexed with i. Messages received by S will have a "hat" and are indexed with j. Finally, messages received by C will have a "hat" and are indexed with i. For example, a random string sent by instance S_j will be denoted by r_j. Upon delivery to instance C_i (potentially the same) string will be denoted $\hat{r}_i$.

1. *Adv creates a bad client B^i:*
   No action needed.

2. *Adv creates (the only) honest client C:*
   No action needed.

3. *Adv creates an instance $S_j^{B^i}$ of S and starts the protocol (recall, B^i is corrupt):*
   $Dist$ follows the protocol.

4. *Adv creates an instance S_j^C of S and starts the protocol:*
   $Dist$ generates the protocol messages by querying O_F. We need this so as to be able to exploit the win of Adv and translate it into a win of $Dist$. Therefore, $Dist$ chooses a random $r_j \in_R \{0,1\}^n$ and queries $O_F(0,2,r_j)$. (Our convention is to index the calls to the long-term key with 0.) He then sends $(r_j, O_F(0,2,r_j))$.

5. *Adv creates new (i-th) instance C_i of the honest client C.*
   No action needed from $Dist$.

6. *Adv delivers a message $m_{C_i} = (\hat{r}_i, \widehat{\mathsf{MAC}}_i)$ to an instance C_i of honest client C (allegedly) from server S:*
   $Dist$ verifies MAC as follows. If $Dist$ ever generated and sent the pair $(r_j, \mathsf{MAC}_j) = (\hat{r}_i, \widehat{\mathsf{MAC}}_i)$ (this could only have been done in Step where an instance of S was created), then we consider MAC verified. Otherwise, $Dist$ emulates failure of MAC verification. Note, by PRFG properties, this action (failure emulation) is indistinguishable from real execution, and a polytime Adv would not detect a deviation. If MAC was verified, $Dist$ continues, and chooses $r_i \in_R \{0,1\}^n$. Then $Dist$ calls $O_F(r_j,3,r_i)$ (recall, $\hat{r}_i = r_j$), and sends $r_i, O_F(r_j,3,r_i)$. According to the protocol, $Dist$ outputs (sk, sid), where $sk = O_F(r_j,4,r_i)$ and $sid = (r_j, r_i)$. However, Adv is only given sid at this time, so $Dist$ does not make the above oracle call, but simply gives sid to Adv. We also note that in the real execution, the key KD is derived from the long-term key, while in our emulation, we use a random key. This difference, however, cannot be exploited by a polytime Adv, due to the PRFG properties of F.

7. *Adv delivers a message $m_{S_j} = (\hat{r}_j, \hat{f}_j)$ to S_j^C (allegedly) from client C (recall, C is honest, and $(r_j, O_F(0,2,r_j)$ is the message previously sent by S_j^C):*
   $Dist$ verifies the message similarly to the verification of the message received by C_i, as follows. $Dist$ accepts the message only if the following holds. $Dist$ must have generated and sent the pair $(\hat{r}_j, \hat{f}_j)$ in the Step where a message m_i was delivered to an instance C_i. Further, this message m_i must have contained the randomness

r_j generated and output earlier by S_j. If either of the above two conditions does not hold, *Dist* emulates failure of MAC verification. Note, by PRFG properties (a hybrid argument), this action is indistinguishable from real execution, and a polytime *Adv* would not detect a deviation. Now we return to the case when verification succeeds. According to the protocol, *Dist* outputs (sk, sid), where $sk = O_F(r_j, 4, r_i)$ and $sid = (r_j, r_i)$ (recall, $\hat{r}_j = r_i$). However, *Adv* is only given *sid* at this time, so *Dist* does not make the above oracle call, but simply gives *sid* to *Adv*.

8. *Adv delivers a message m_{S_j} to $S_j^{B^i}$ (allegedly) from client B^i(recall, B^i is corrupt):*
   Dist emulates protocols execution according to specification, and responds with the corresponding output instruction. Recall, in dealing with corrupt clients, *Dist* does not use the oracles of the game G, but simply fully follows the protocol.

9. *Adv sends an* open *request on a (completed and not failed or challenged) client instance C_i of C:*
   Recall that C_i output $sid_i = (\hat{r}_i, r_i)$. *Dist* queries oracle $O_F(\hat{r}_i, 4, r_i)$, and gives the answer to *Adv*. Note that there are restrictions on when *Dist* is allowed to call O_F (O_F and O_C cannot be called with the same parameters). We argue later that we are not violating them.

10. *Adv sends an* open *request on a (completed and not failed or challenged) server instance S_j of S:*
    Recall that S_j output $sid_j = (r_j, \hat{r}_j)$. *Dist* queries oracle $O_F(r_j, 4, \hat{r}_j)$, and gives the answer to *Adv*. As in 9, we will later argue that we are not violating G's restrictions.

11. *Adv sends a* challenge *request on a (completed and not failed or opened) server instance S_j^C of S:*
    Recall, S_j^C sent r_j, received $\hat{r}_j$ and output $sid_j = (r_j, \hat{r}_j)$. *Dist* queries the challenge oracle $ch = O_C(r_j, \hat{r}_j)$, gives ch to *Adv* and submits *Adv*'s output as his answer to the challenge of G. As in 9 and 10, we will later argue that we are not violating G's restrictions when querying $O_C(r_j, \hat{r}_j)$.

12. *Adv sends a* challenge *request on a (completed and not failed or opened) client instance C_i^S of C:*
    This is handled symmetrically to the above request 11. *Dist* queries the challenge oracle $ch = O_C(\hat{r}_i, r_i)$, gives ch to *Adv* and submits *Adv*'s output as his answer to the challenge of G. Again, we will later argue that we are not violating G's restrictions when querying $O_C(\hat{r}_i, r_i)$.

We now argue that all calls to O_F in 9–10, and to O_C in 11–12 will be legal requests in G, that is that *Dist* never calls both $O_C(i, r)$ and $O_F(i, 4, r)$ for any pair (i, r).

First note that O_F and O_C are only called when *Adv* opens or challenges instances, respectively. Suppose, *Adv* challenged a server instance S_j^C, and thus caused the call $O_C(r_j, \hat{r}_j)$, where $\hat{r}_j = r_i$ was generated by client C_i. We now have to show that $O_F(r_j, 4, r_i)$ was not and will not be called. Consider two possible cases. First, for $k \neq j$, *Adv* opens (either earlier or later) a server instance S_k, causing a call $O_F(r_k, 4, r_i)$. This is not a problem, since $Prob(r_j = r_k)$ is negligible. Second, Adv opens a client instance C_k^S, thus causing a call $O_F(r_m, 4, r_k)$. Suppose this call is illegal, i.e. $r_m = r_j$ and $r_i = r_k$. However, in this case, the session ids output by the parties match. Then S_j^C and C_k^S are partners, and such Adv's behavior is not allowed in KE.

By assumption of the lemma, Adv wins with probability non-negligibly more than 1/2. It is easy to see that $Dist$ wins whenever Adv wins, except for a negligible fraction of the time. Therefore, the constructed $Dist$ wins the game G with probability non-negligibly more than 1/2.

□

Lemma 2 *If the PRFG F used in G is secure, then there does not exist a polytime $Dist$ winning the game G with probability $p > 1/2 + \delta$, where δ is not negligibly small (in the security parameter n).*

The proof of Lemma 2 is done by a standard hybrid argument, and is omitted.

□